Dedicated multiplier ICs speed-up processing in fast computer systems

Increasing computation speed is not only important, but crucial to the successful development of certain computers—notably real-time digital signal processors and scientific data processors. In these machines, there is very limited time available to carry out a calculation. And multiplying operations can eat up a lot of that time.

But now you can save time because the availability of dedicated multiplier ICs lets you reduce multiplication operations from microseconds to the 100-ns range. Of course, you can always multiply with repeated add-and-shift operations. But suppose you have a speech processing system whose data are sampled at an 8-kHz rate (every 125 μs). If your processor has to compute, say, an auto-correlation function for 256 samples of a speech waveform, it will have to multiply each sample by 256 other samples —all in 125 μs. In other words, every operation must be completed in less than 500 ns. Only a dedicated multiplier can do that.

A dedicated chip is certainly faster than conventional methods, but it's also more expensive. So before you decide to include multiplier ICs in a system, first consider how often the system must multiply.

Are multipliers for you?

Traditional computer multiplication, by the add-and-shift method, produces the product of two N-bit numbers in n cycles. So multiplication in a 16-bit system, takes 16 times longer than addition. In a typical business application, the computer may use the multiplication function in just 6% of all instructions. In other words, out of every 100 instructions, 94 are additions and six are multiplications. A simple calculation will help you decide whether a dedicated multiplier significantly improves your system throughput speed.

Assume that addition takes one instruction cycle. Add-and-shift multiplication in a 16-bit system will require 16 cycles. Now, to execute 100 instructions it takes

$$(94 \times 1) + (6 \times 16) = 190 \text{ cycles}$$

But a dedicated chip can multiply in the same time that it takes to add; so 100 instructions will be executed in

$$(94 \times 1) + (6 \times 1) = 100 \text{ cycles.}$$

That means an improvement of

$$190/100, \text{ or } 1.9.$$

An improvement of almost two times isn't bad, but you'll have to decide whether it's cost-effective for you.

On the other hand, a digital signal processor might have to multiply half the time. Assuming that add and multiply times are the same as for the business application, you'll get a larger improvement factor with dedicated multipliers:

$$\frac{(50 \times 1) + (50 \times 16)}{(50 \times 1) + (50 \times 1)} = \frac{850}{100} = 8.5$$

Increasing speed performance more than eight times, probably justifies going with dedicated multipliers.

Although speed is the most important characteristic of a multiplier chip, there are other factors that you must evaluate in order to select the proper device for your system.

Power is a key characteristic because excessive dissipation in the chip may force you to design a special cooling arrangement to avoid damaging components (see "A power refresher").

Word length depends largely on your system requirements. Available multipliers usually offer operand lengths of four and eight bits, but some even offer 16-bit capability. Also keep in mind that it's possible to cascade some devices to expand their bit capacity.

Data representation can help you narrow the selection of a chip. Some devices allow only two's-complement data representation, while others handle only unsigned data. But the most powerful chips let you handle both.

Secondary factors include packaging (20-pin Slim vs 64-pin DIP with heat sink), rounding ability and price.

All multipliers can be grouped into two broad speed categories: high-speed parallel and the (slower) sequential types.

Fast and faster

How multiplication is performed on the chip differs for each type. Sequential devices produce partial products step-by-step, while parallel types generate

Shlomo Waser, Product Planning Manager, Monolithic Memories, Inc., 1165 E. Arques Ave., Sunnyvale, CA 94086.

Reprinted from Electronic Design-September 13, 1978

Compare multiplier characteristics

SPEED CATEGORY	DEVICE	CONFIG-URATION	PKG PINS	MAX SPEED (ns)	MAX POWER (WATTS)	DIVIDE	DATA REP
HIGH-SPEED PARALLEL MULTIPLIERS	MMI 67558	8 × 8	40	125	1.4	NO	2's COMP AND UNSIGNED
	TRW MPY-8	8 × 8	40	170	1.5	NO	TWO's COMPLEMENT
	TRW MPY-12	12 × 12	64	200	3.75	NO	
	TRW MPY-16	16 × 16	64	230	5	NO	
	AMD 25S05	2 × 4	24	40	0.9	NO	2's COMP
	MOT 10183	2 × 4	24	25	1.0	NO	2's COMP
	TI 74S274 (ROM)	4 × 4	20	70	0.6	NO	UNSIGNED
MEDIUM SPEED SEQUENTIAL MULTIPLIERS	AMD 25LS14	8 × 1	16	50	0.6	NO	2's COMP
	AMD 25LS-2516	8 × 8	40	400	1.4	NO	2's COMP
	MMI 67516	16 × 16	24	800	1.3	YES	2's COMP
	MMI 67508	8 × 8	20	400	0.75	YES	

A power refresher

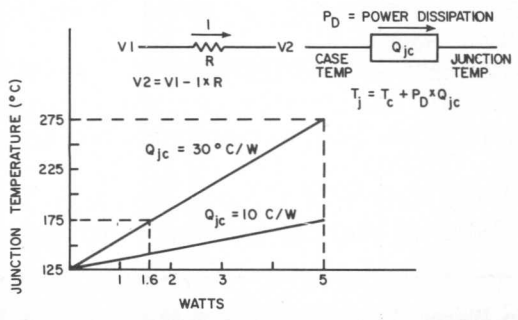
Excessive power dissipation in semiconductor devices causes junction temperatures to rise, and eventually leads to failure. Junction temperature should never exceed 175 C. You can calculate junction temperature from the following well-known equation,

$$T_j = T_c + P_d \times Q_{jc}$$

where T_j is the junction temperature
T_c is the case temperature of the IC
P_d is the power dissipation in the IC
Q_{jc} is the thermal resistance from junction to case.

This relationship is similar to Ohm's law except that the variables represent temperature, power and thermal resistance instead of voltage, current and electrical resistance.

In the following graph, junction temperature is shown as a function of power dissipation (military specs limit maximum case temperature to 125 C). The line on the graph marked Q_{jc} = 30 C/W represents thermal resistance of a typical LSI package such as a 40-pin ceramic DIP. At a dissipation of 5 W, you can see that junction temperature will be 275 C, which will ensure destruction of the IC. To avoid this, you can cool with forced air and glue a heat sink to the package, which then puts you on the Q_{jc} = 10 C/W line. Now you can dissipate 5 W without damaging your IC. If you can't use special cooling techniques, you must limit maximum power dissipation to 1.6 W.

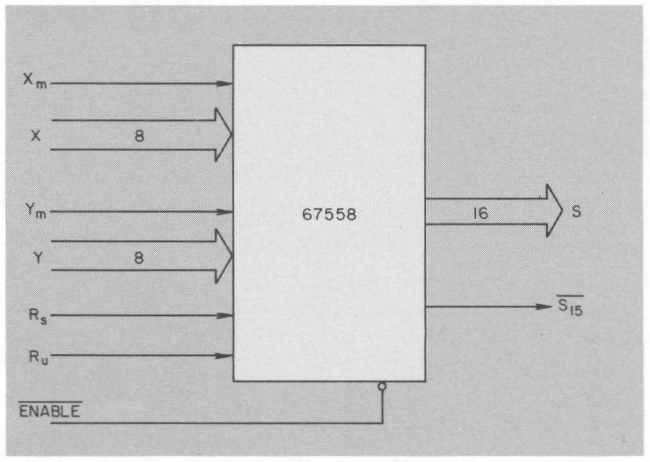


them in one step.

In one parallel multiplier, a matrix of partial products is generated and then reduced with pseudo-adders. Another design uses a collection of iterative cells in an array, where partial products are generated in one step and reduced afterwards. Top speeds of small parallel devices (2 × 4 configuration) are about 24 ns, while the big 16 × 16 chips do a multiplication in around 250 ns.

Sequential multipliers, which go as slow as 1 μs for 16 × 16 chips, are constructed with time-sequenced logic. Operands are loaded into on-chip working registers, one operand during each clock period, under the sequencer control. When two operands have completed loading, the device jumps to the multiply routine, which requires several clock periods. The resulting product is then ready to be placed on a bus, in a time sequence.

Now you're ready to take a look at some of the parallel and sequential devices that are on the market.



1. **A 16-bit product is generated** by this 8 × 8 multiplier, the Monolithic Memories 67558. You can feed your input data to the X and Y terminals in either two's complement or unsigned format.

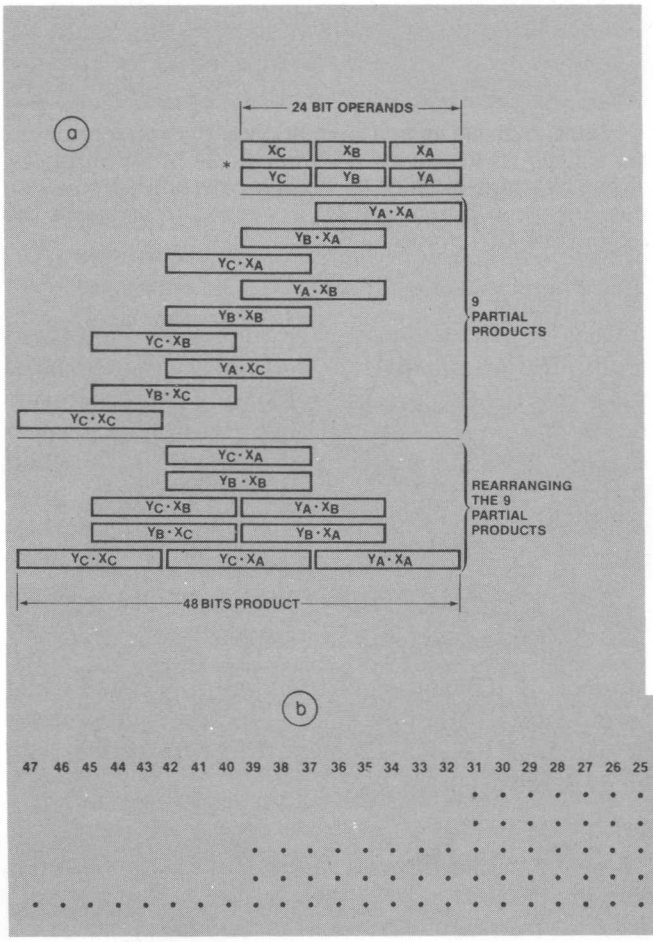
Products that form products

Most commercially available multipliers are listed with their key specifications in the Table, "Compare multiplier characteristics." The AMD25S05 (Advanced Micro Devices) was the standard building-block in the early 70's in the design of many high-speed processors, and uses the iterative cells in an array technique. It forms the mathematical function XY + K, where K is an input fed by the partial product from an earlier stage in multiplication.

A single 2 × 4 25S05 multiplies in 25 ns typical, but to multiply two 16-bit words you'll need 32 of them. Delay time then jumps to 150 ns. Motorola's 10183, another 2 × 4 chip, is an ECL device than can be cascaded to multiply two 16-bit words in 100 ns. TRW's MPY-16, a 16-bit device, is aimed at digital signal processing with a so-called "pipe-line" design, which means that the chip's input and output registers are clocked simultaneously.

The only multiplier in the table that accepts both two's complement and unsigned data is Monolithic Memories' 67558. Housed in a standard 40-pin DIP, the 8 × 8 device (see Fig. 1) has a maximum delay of 125 ns, dissipates only 1 W and requires no special cooling. Eight-bit operands are fed into the X and Y inputs, while inputs X_m and Y_m define the type of data for each operand. The R_s and R_u lines are used for rounding operations. The 67558's output product is a 16-bit signal (shown as S in Fig. 1).

In many applications of dedicated multipliers, you'll find it necessary to expand their capacity to handle larger words. The 67558 is expandable to N × N. Say you want to multiply two 24-bit operands to produce



2. **Multiplying two 24-bit operands together** takes two steps. In the first one (a), nine partial products are generated, which must be added to produce the final result. Rearranging the partial products in the second step is represented by the series of dots (b), which tell you the number of factors in each partial product. Even though nine partial products are produced, they are treated as five operands.

a 48-bit result. Internally, the chip takes two steps to do it (Fig. 2a). First, partial products are generated; second, it adds them together to produce the final result. Multiplying two 24-bit operands produces nine partial products (3 eight-bit operands × 3 eight-bit operands). You can find the number of multipliers (n × m) needed to perform (N × M) multiplication with the following equation:

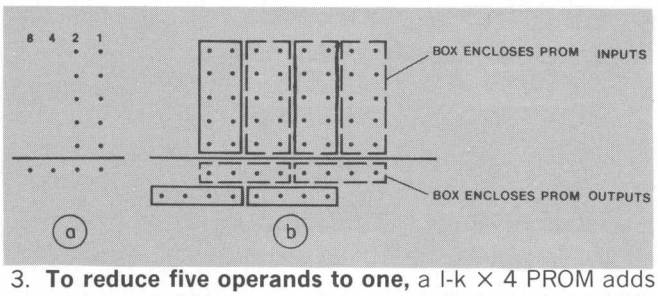
$$\text{No. of devices} = (N/n) \times (M/m).$$

To do a 24 × 24 multiplication with 4 × 4 devices, you would need

$$(24/4) \times (24/4) = 36 \text{ multipliers.}$$

The addition of partial products is shown by the dot representation in Fig. 2b. Although nine partial products are generated, they represent only five operands. However, all standard adders accept just two input operands, so you need a method of reduction. And the method is called column reduction—n operands are reduced to two, which are then added by carry-lookahead adders. For a 24 × 24 multiplication, a 1k × 4 PROM reduces five operands to one (Fig. 3). The PROM treats 10 address lines as two adjacent columns of five bits each, and performs a table look-up to find the sum. Note in Fig. 3 that the maximum sum is 15, which requires exactly four bits for its binary representation.[1]

Column reduction of a 24-bit multiplication is shown in Fig. 4. To reduce five operands to one in this case, you need twelve 1k × 4 PROMs, ten 74S381s and two 74S182s to perform carry-lookahead addition on the operands. The worst case time to complete the multiplication is 262 ns—125 ns for the 8 × 8 multipliers, 50 ns for the PROMs and 87 ns to perform the carry-lookahead addition.



3. **To reduce five operands to one,** a I-k × 4 PROM adds two columns of binary numbers, each 5 bits high (a). This produces a 4-bit output whose maximum value is 15, which can be represented exactly by four binary bits—the number of output lines on the PROM (b).

Having found the 48-bit product of two 24-bit numbers, you must reduce the double-length product (48 bits) to a single-length product (24 bits). And two techniques are used in data processing systems to do this—rounding and truncating.

Rounding-truncation tradeoffs

Rounding is the preferred reduction method when precision is your main consideration. The following results of rounding and truncating decimal numbers will show you why.

$$\left.\begin{aligned} 39.2 &\rightarrow 39 \\ 39.6 &\rightarrow 39 \end{aligned}\right] \text{ truncating}$$

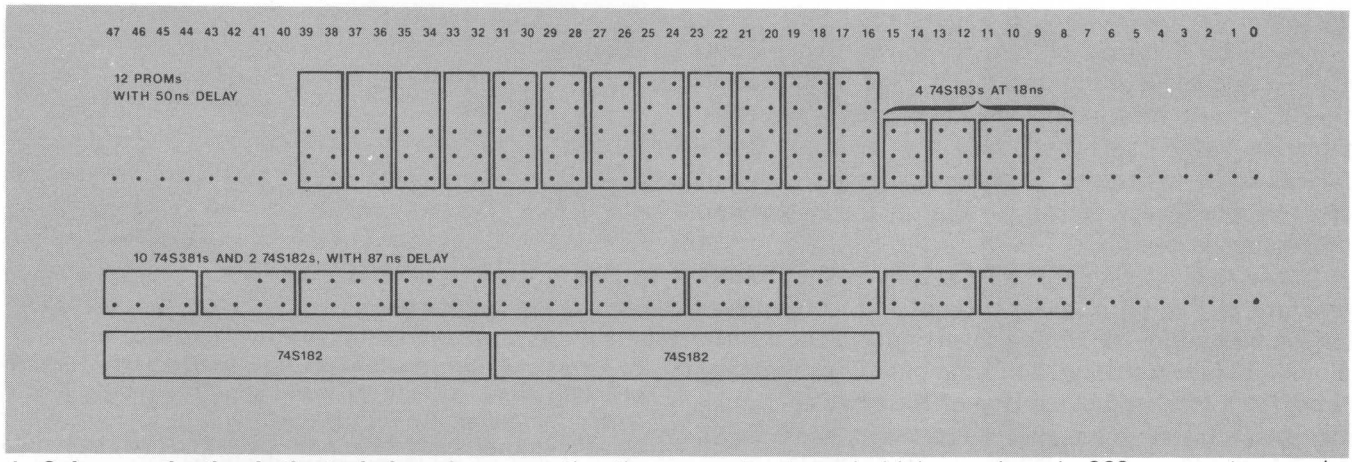
$$\left.\begin{aligned} 39.2 + 0.5 &= 39.7 \rightarrow 39 \\ 39.6 + 0.5 &= 40.1 \rightarrow 40 \end{aligned}\right] \text{ rounding}$$

However, rounding requires an extra step—adding one-half the weight of the single-length LSB to the MSB of the discarded part. To round a number like 39.28 to one decimal point you'll have to add 0.05 to the number, then truncate the LSB.

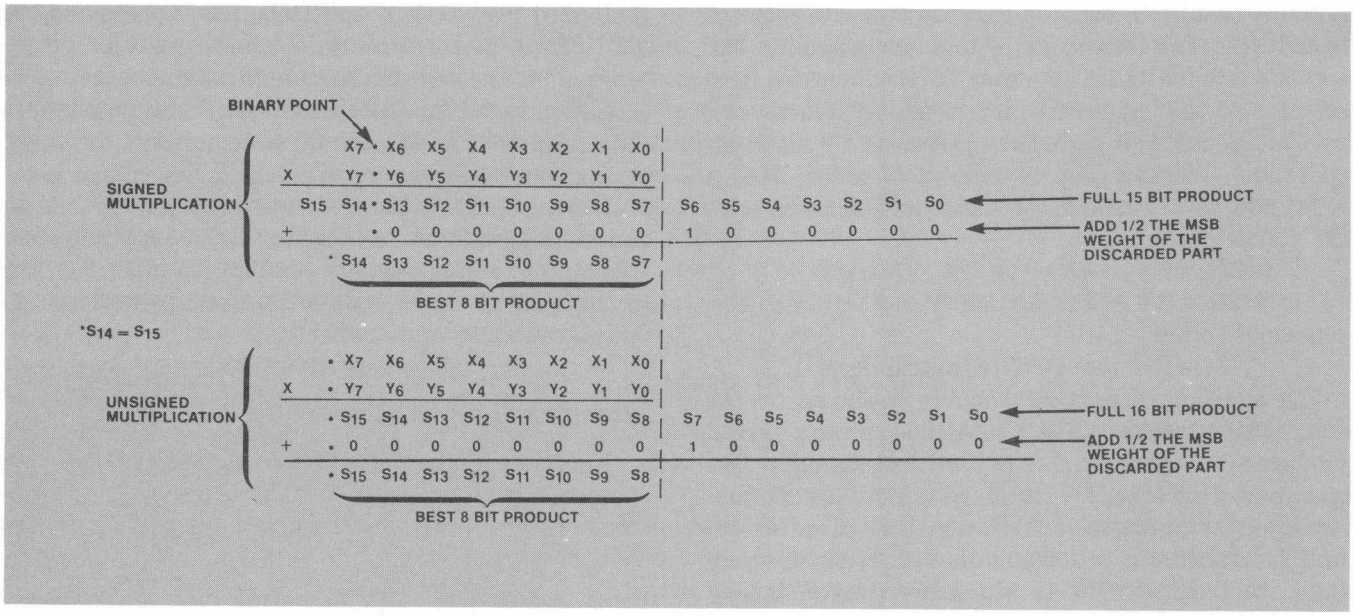
$$39.28 + 0.05 = 39.33 \rightarrow 39.3$$

Rounding in binary arithmetic is similar to rounding in decimal except that you have to deal with signed and unsigned data. In signed multiplication the two MSBs are identical for all practical purposes. Therefore you get the most precise single-length product when the result is shifted one bit to the right. A look at Fig. 5 will show you how signed and unsigned multiplications are rounded to get the best final product. For signed data, add one-half the S_7 bit-weight by adding a ONE to bit position 6; for unsigned data add a ONE to bit position 7.

With a 67558, rounding is handled via the R_s and R_u input lines. If you want a rounded single-length



4. **Column-reduction is the technique** for generating the result of two 24-bit numbers multiplied together. You'll get extremely high-speed—only 262 ns worst case—but it takes a lot of hardware to get the job done.



5. **Rounding the results of both two's complement and unsigned multiplications** gives you more precise data than you'll get by truncating. To round a two's complement number, add a ONE to S_6 for the best result. For the best unsigned number add a ONE to S_7. The best 8-bit product is the most significant half of the product.

result for signed data, tie the R_s line to V_{cc} and ground the R_u line (vice-versa for unsigned data). For a double-length product, ground both R lines and no rounding will take place.

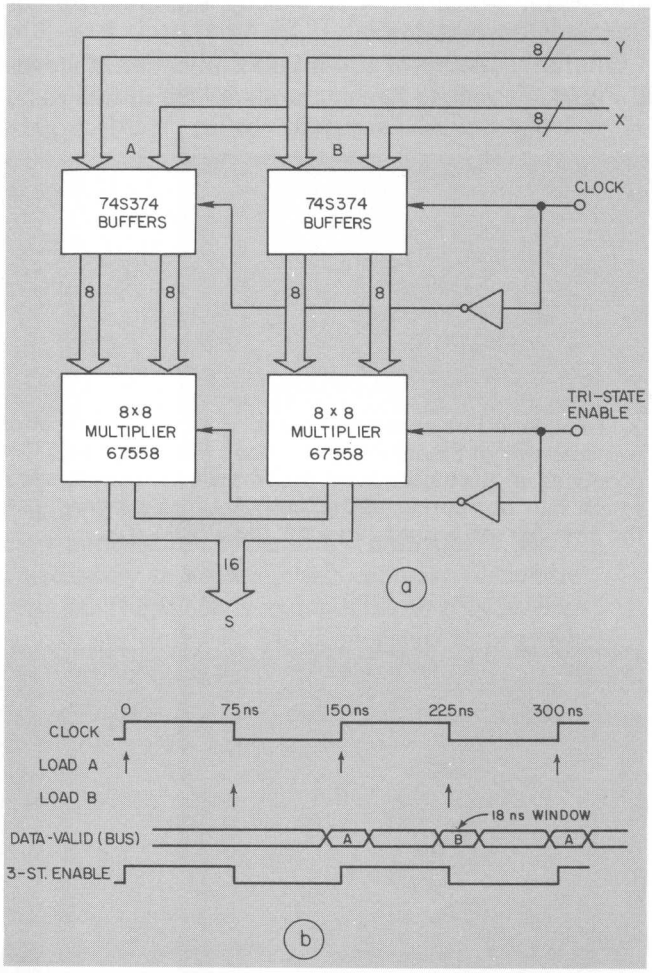
Rounding may be preferable sometimes, but it slows down multiplication. There will be other applications where you'll not only want to avoid a slowdown, you'll need a technique that allows you to improve the speed of a multiplier beyond its data sheet specifications.

Getting more speed

As the table shows, a 67558 can multiply two 8-bit numbers with a maximum delay of 125 ns. But if your application requires more speed, you'll have to use a technique called overlapping, better known as a "ping-pong" operation. Ping-ponging nearly doubles system throughput, but it takes two 8 × 8 devices to perform an 8 × 8 multiplication. Actually, using three or more devices gives you even greater speed, but an example with two devices is easy to understand.

The block diagram of Fig. 6a shows two 67558s connected for speed enhancement. Ping-pong means they'll operate alternately—one performing a multiplication while the other loads the next two 8-bit factors to be multiplied. A three-state enable signal controls placement of the results on the bus when processing is complete.

To see how overlapping works, look at the timing diagram of Fig. 6b. A clock cycle of 150 ns has been chosen because of the timing constraints of the multipliers—125 ns multiplication time plus 17 ns register time, for a total of 142 ns. If your first two operands are placed into input register A at t = 0, you'll have a product 142 ns later. While the A multiplier performs this multiplication, the input bus feeds two new



6. **Ping-pong two multipliers to speed-up your multiplication time.** Instead of working with a single 8 × 8 device, ping-ponging with two lets one of them multiply while the other loads data. And speed goes way up—data appear on the output bus at 75-ns intervals, rather than 142 ns, as with one device.

8-bit operands into the register at t = 75 ns. And 217 ns from time zero, the results of the B computation are put onto the bus. (75 + 142 = 217). However, at t = 150 ns, or 8 ns after A finishes computing, new input data are loaded into the A multiplier. And the A output will be ready after 292 ns (150 + 142), but notice that the output lines receive data spaced at about 75-ns intervals. So, in effect, multiplication time is 75 ns—little more than half the time it would take a single 67558. Add a third multiplier and you'll reduce the 75 ns down to 50 ns. But remember that you'll also need more enable logic.

Incidentally, data are available on the output lines for only about 18 ns. Make sure that the logic accepting this information is capable of capturing and holding it within that 18-ns window.

Pipelining will also speed up a 67558 or any other dedicated multiplier. But you can only use it where expansion of multipliers is required, because it involves capturing partial products, then pipelining them to adders. To make pipelining practical, put it to work after you've previously expanded multipliers, and your time delays between operand input and partial product output will be nearly identical to the time required for column reduction. But this method can't be used in a single-multiplier design problem since partial products are never available—they're handled internally.

Since dedicated chips can multiply at high speed, it's natural to wonder if they can rapidly perform division, another time-consuming operation. And the answer is yes, because division is simply an extension of multiplication.

Divide with reciprocals

More specifically, division is the process of multiplying one number by the reciprocal of another:

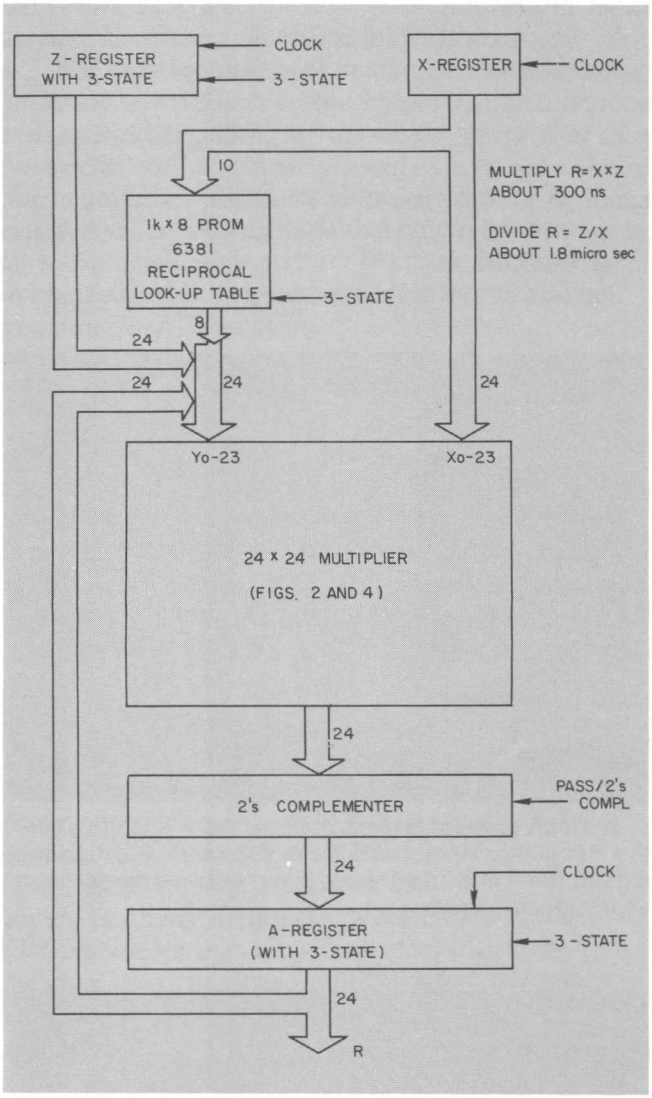
$$a \div b = a \times 1/b.$$

Therefore there's nothing very difficult about implementing high-speed division with dedicated multipliers. One method is to make a 256 × 8 ($2^8 \times 8$) PROM serve as a reciprocal look-up table. Its output then becomes one of the operand inputs to the multiplier.

But suppose you want to multiply two 24-bit numbers together with the PROM. You'd need a look-up table of reciprocals that uses 48-megabytes ($2^{24} \times 24$) of ROM! That's not only impractical, but virtually impossible.

To solve the problem, a scheme to produce reciprocals similar to that found in the IBM 360 is required[2]. This algorithm begins by calculating the reciprocal of an 8-bit number, then deriving the reciprocal of a 16-bit number. This leads to the reciprocal of a 32-bit number—stop when you reach your desired number size. The process assumes a normalized number—one with a characteristic between 0.5 and 1.0.

The hardware you need to perform 24-bit division is shown in block-diagram form in Fig.7. Note that part of this division circuit already contains a 24 × 24 multiplier.



7. **You can divide as well as multiply** with dedicated chips. In fact, you'll first need a complete 24 × 24 multiplier circuit (shaded block) to build a division circuit like this one. In addition, a ROM is needed to generate the reciprocals of numbers.

Once you've established techniques for high-speed multiplication and division, you can use them to build a low-cost floating processor. Floating-point representation of numbers is better than fixed-point to cover a large dynamic range. For example, in a 32-bit system, fixed-point representation (integers) gives you a range of only 1 to 10^{10} (or 2^{32}). But with floating-point those 32 bits span the range of 10^{-79} to 10^{+75}.

With about 70 ICs such a processor can add, subtract and multiply in 400 ns, and divide in 1.6 μs. Couple this high performance with the relatively small number of ICs and significant improvements could occur in minicomputers, since a floating-point processor will then become a standard feature rather than an expensive option. ■■

References

1. Stenzel, W. J., et al, "A Compact High-Speed Multiplication Scheme," *IEEE Transactions on Computers*, Vol. C-26, No. 10, Oct. 1977, pp. 948-957.
2. Anderson, F. S., et al, "The IBM System 360/91 Floating Point Execution Unit," *IBM JRD*, Vol. 11 #1, Jan. 1967, pp. 34-53.